AIAA-2001-2584

Application of a Scalable, Parallel, Unstructured-Grid-Based Navier-Stokes Solver

Paresh Parikh
NASA Langley Research Center
Hampton, VA

15th AIAA Computational Fluid Dynamics Conference
June 11-14, 2001/Anaheim, California

APPLICATION OF A SCALABLE, PARALLEL, UNSTRUCTURED-GRID-BASED NAVIER-STOKES SOLVER

Paresh Parikh*
NASA Langley Research Center
Hampton, Virginia, 23681

ABSTRACT

A parallel version of an unstructured-grid based Navier-Stokes solver, USM3Dns, previously developed for efficient operation on a variety of parallel computers, has been enhanced to incorporate upgrades made to the serial version. The resultant parallel code has been extensively tested on a variety of problems of aerospace interest and on two sets of parallel computers to understand and document its characteristics. An innovative grid renumbering construct and use of non-blocking communication are shown to produce super-linear computing performance. Preliminary results from parallelization of a recently introduced "porous surface" boundary condition are also presented.

INTRODUCTION

As computers gain in speed and efficiency, rapid Computational Fluid Dynamic (CFD) simulations using Navier-Stokes (N-S) equations is becoming a design requirement. Once a very difficult proposition, full N-S solutions in a short (1-2 week) time, have become attainable in recent years due to use of unstructured grids. Over the past few years, it has become evident that the turn-around time can be further reduced by the use of parallel N-S solvers on multi-CPU computers. Parallel computing involves distributing the computational load to a number of processors for simultaneous operations. An obvious advantage of parallel computing is the reduced wall-clock time resulting in rapid turn-around. The other, not-so-obvious, advantages include use of a large number of smaller CPUs instead of a large super-computer and off-hour utilization of computational resources, which would otherwise idle during these hours. Several parallel, unstructured grid-based Navier-Stokes solvers have been developed and recently reported in the literature. Examples can be found in Refs. 1-3.

This paper describes parallelization of the Navier-Stokes solver **USM3Dns**, belonging to the TetrUSS software system. **TetrUSS** is a tetrahedral, unstructured grid-based complete flow analysis system for CFD analysis of complex configurations. The system consists of components for grid generation, flow solution and pre- and post-processing. An overview of all the components of TetrUSS and their capabilities can be found in reference [4].

Parallelization of an earlier serial version of USM3Dns was reported in Ref. 1. Since then an improved version of the serial code has been developed. These improvements include revised vorticity computations for more accurate turbulent production terms, a different restart file format, enhanced user functionality related to debugging and diagnostics, and introduction of a "porous surface" boundary condition (BC) [5].

The main aim objectives of the of the present work include, (1) upgrade of the previously developed parallel version to include the latest improvements made to the serial code, using techniques reported in Reference 1, (2) evaluation of the efficiency of the resultant parallel code on a variety of parallel environments, and (3) do so without degrading the accuracy as compared to the serial code.

In the following sections, salient features of the flow solver are first briefly described, followed by an overview of the main features of the parallel code. Finally, the performance and efficiency of the parallel solver is demonstrated on three configurations.

---

*Senior Aerospace Engineer, Associate Fellow AIAA

N-S SOLVER USM3Dns

The flow-solver component of **TetrUSS** is called USM3Dns[6], which is a three-dimensional, tetrahedral, cell-centered, finite-volume Navier-Stokes solver. Inviscid flux quantities are computed across each cell face using Roe's flux-difference splitting (FDS) [7]. Spatial discretization is accomplished by a novel reconstruction process, which is based on an analytical formulation for computing solution gradients within tetrahedral cells. The solution is advanced to a steady state by an implicit backward Euler time-stepping scheme. Flow turbulence effects are modeled by the Spalart-Allmaras (S-A) one-equation model [8]. In addition to full viscous solution capability, a wall-function formulation is also available to reduce number of cells in the boundary layer.

The flow solver and its companion programs in TetrUSS are widely used by many in the US aerospace industry, academia and Government research laboratories.

Parallelization Features

Since fluid dynamic computations are grid based, parallelization is achieved by dividing the computational grid into a large number of smaller zones (partitions) and assigning each zone to a CPU. All CPUs then simultaneously work on different part of a computational grid, sharing information when necessary and thus advancing the flow solution. The two main elements for an efficient parallel code are load balancing and efficient communication among processors. Load balancing assures an equitable work distribution to all CPUs involved. An efficient communication between processor is important because data sharing between neighboring cells and points, which may be physically assigned to different CPUs, is needed many times during the flow solution iteration process.

Grid Partitioning

A partitioning algorithm is required to form suitable grid partitions for parallel operation. For optimum performance, the grid partitions not only need be of nearly equal size, but also should be physically contiguous and should have the smallest possible inter-partition boundary to minimize communication requirements.

As reported in Ref. 1, grid partitioning is achieved by a mathematically rigorous "multi-level spectral bifurcation" graph partitioning technique called Metis [9]. The basic Metis algorithm divides the grid in groups of contiguous points or nodes. Since the grid used in TetrUSS needs to be divided in groups of contiguous tetrahedral cells, the Metis program has been modified to be carryout this function. The partitioning algorithm is very efficient and only requires a few seconds of CPU time to divide a grid into a user specified number of "nearly equal" partitions.

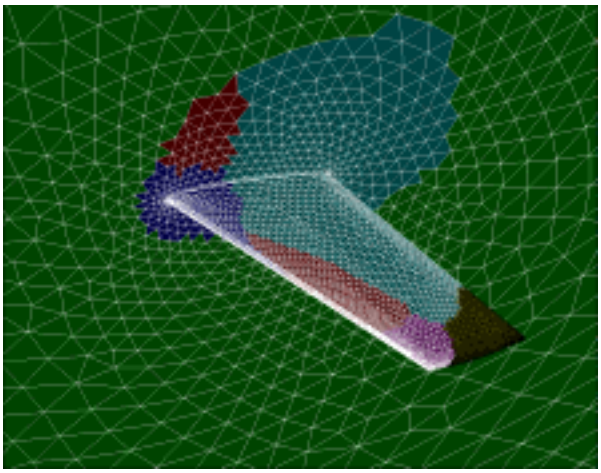

Figure 1 - Grid on an ONERA M6 wing divided into 6 nearly equal partitions.

Figure 1 shows a viscous, tetrahedral unstructured grid on an ONERA M6 wing configuration, partitioned into six (6) zones, each shown in a different color. The original grid has 495,012 cells. The partitions vary in size from 78,679 to 84,977 cells. Compared to a perfectly balanced size of 82,502 cells per partition, the partitioning results in about a +3% to –4.6% imbalance.

Grid partitioning is accomplished as an independent pre-processing step and provides the solver with a partition number associated with each computational cell. The solver uses this information to assign appropriate grid geometry data to each CPU.

Parallelization of the Solver

The present parallel effort is based on the techniques reported in Reference 1. Accordingly, all the modifications to the serial code have been accomplished by embedding suitable Message Passing Interface (MPI) calls at appropriate places in the code. Salient features for some of these

operations are briefly described in this section. For details, the reader is referred to Ref. 1.

Input/Output (I/O)

In an effort to keep the code portable and able to run in a variety of parallel environments, all the required I/O is conducted in the root (or master) processor. The global grid and cell-to-partition correspondence is read by the root processor and broadcast to other processors. Using this data, each processor performs the following tasks:

1. Extracts data needed for its own use and renumbers the local grid, and
2. Finds the grid cells, nodes and faces that lie on the inter-partition boundaries through which inter-processor communication will take place during flow solver iterations.

Upon program completion, each processor sends its own data to the root processor, which properly assembles the data and writes all output files, including the restart file. Such an operation affords true scalability in that all the output files are 'universal', i.e. a restart file written at the end of a run with "n" processors can be read in a subsequent run using "m" processors. However, it may be noted that the partitioning program needs to be re-run for changing the solution from "n" to "m" processors.

Data Structures

The flow solver employs a cell centered, upwind based scheme, and the information flows from a cell to its neighboring cells through cell faces. Thus, cell faces on the partition boundaries of a partition need to communicate with cells, which physically reside in another CPU. Inter-partition cells outside a partition are termed as partition ghost cells and their non-inter-partition node as ghost nodes. Updating of flow variables for partition cells on the inter-partition boundaries requires current flow variable values at ghost cells and nodes. This is accomplished by specialized communication send- and receive-arrays between a partition and its neighbors. These 1-D arrays are formed as a pre-processor step and values contained in these arrays are constantly updated as the flow solution progresses. All data structures, being memory overhead, are optimized for efficiency and form an insignificant part of the overall required memory.

Iteration Loop

The implicit formulation of USM3Dns leads to a sparse set of equations that are iteratively solved using a Gauss-Seidel (GS) type iterative procedure. One GS cycle is achieved by doing up to 20 sub-iterations; each using updated values from the previous sub-iteration. At the end of each sub-iteration, communication is required to update data corresponding to ghost cells/nodes. Such a communication overhead presents serious performance degradation. To overcome this "bottleneck", an innovative concept involving overlapping of computations and communications is devised. Accordingly, cells in each partition are renumbered beginning with the cells on the partition interface boundaries, followed by interior cells and finally the partition ghost cells. At the end of the update of flow values at the partition interface cells, while these values are communicated to the appropriate places, the interior cells are simultaneously updated, thereby overlapping the computation and communication times. For the most part, this results in little or no communications overhead. Use of this specialized technique has greatly improved the time per iteration performance of the parallel code.

Parallelization of Porous Boundary Condition

A boundary condition suitable for simulating passively porous aerodynamic surfaces has been recently added to the serial version of the flow solver USM3Dns[5]. The implementation assumes a constant pressure plenum underneath a porous surface. The plenum pressure is a function of the net mass flux over the entire porous surface and is determined iteratively. Parallel implementation of the porous BC requires:

1. transfer of mass flux from CPUs containing porous surfaces to the root processor,
2. calculation of "net" mass flux and average plenum pressure, and
3. broadcast of average plenum pressure value to all CPUs for use in the subsequent iteration.

RESULTS

The efficiency and accuracy (in terms of faithful reproduction of serial code data) is demonstrated here by running the parallel and the serial codes on three aerospace configurations. The configurations are a simple ONERA M6 wing configuration and a complex F-16 fighter configuration complete with external fuel tank and a store. Additionally, the validity of the parallelization of the recently added porous BC to USM3Dns is demonstrated on a 5-Caliber Tangent-Ogive forebody configuration.

All of the parallel solutions were obtained on either an SGI Origin 2000 system located at NASA Ames Research Center or a local cluster of Compaq XP1000 Personal Computer (PC) nodes. The Origin 2000 is a shared memory computer, with 256 available nodes (CPUs), each with 256 MB of run time memory and UNIX- based operating system. The PC cluster is a distributed memory system of individual PCs, each with 1GB of main memory and a Linux operating system (Red Hat 6.0). The individual PCs are connected by a high-speed communications bridge.

Customarily, the efficiency of parallel computations is measured by evaluating the utilization of allocated CPUs for a given task. While several ways of measuring parallel efficiency have been reported in the literature, one of the most commonly used measure is the so called, Speed-Up Factor (SF), which is defined as [10]

$$SF = \frac{Baseline\ Processor\ Time/Iteration}{Multi-Processor\ Time/Iteration}$$

The time/iteration is averaged over a number of iterations to smooth out small variations caused by factors such as latency of the computer, available access to memory and computational load. Traditionally, time/iteration on one processor has been used as "Baseline" in the above equation. However, in all the results presented here, a number greater than one is used as "baseline". This is done for at least two reasons; 1) the large grid size for the configuration would not fit on a single processor of any parallel computer for which access was available, and 2) by using a multiple CPU run as the baseline to compare all subsequent multi-CPU runs, the same version of the code was used. If, however, a single CPU run were used as the baseline, differences in the performance of a serial vs. a parallel version of the code also need to be quantified. The Speed-Up Factor, as defined above, has been used in the present study to evaluate efficiency of the parallel code.

ONERA M-6 Wing

The ONERA M6 wing configuration has been traditionally used as a first test case for validation of many new 3-D CFD solvers. An unstructured grid, suitable for full viscous calculations ($Y^+ \cong 1$), with 495,012 tetrahedra and 86,389 points, was generated for this study. The grid was suitably partitioned and run on a number of CPUs for the flow conditions of a free-stream Mach number of 0.8447 and an angle-of-attack of 5.06 degrees.

Figure 2 shows the Speed-Up factor plotted against the number of CPUs. The speed-up factor has been calculated by considering the time per iteration derived from a 10-processor run as "baseline", i.e. assuming it to give a speed-up of 10. Superimposed on the speed-up factors in Figure 2, is a straight-line depicting linear speed-up. As can be seen, the parallel code produces a super-linear performance. This counter intuitive observation has been carefully analyzed and can be explained as follows.

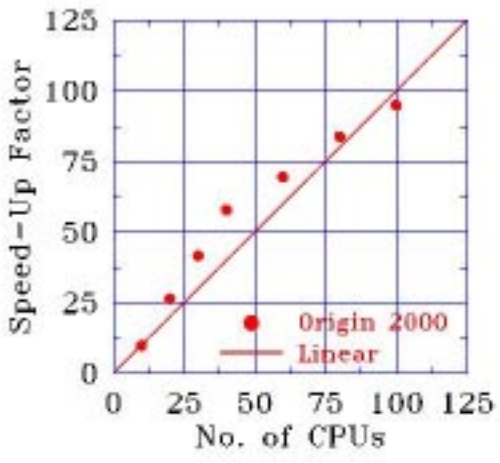

Figure 2 - Speed-Up performance on an Origin 2000 computer system for ONERA M6 Wing. Mach= 0.8774, α = 5.06 deg. Re_c=11.7 X 10^6.

Computations using unstructured grids require access to widely differing array indices due to inherent lack of structure. This requires indiscriminate use of cache, resulting in a large

number of "cache-misses" and hence the time per iteration becomes cache dominated. With increasing number of processors, the grid size assigned to a processor gets smaller and the cache access becomes more efficient, resulting in super-linear performance. Another factor contributing to the super-linear performance is the aforementioned renumbering of the partition grid and the use of non-blocking communication. There is, however, an upper limit to the increasing cache efficiency for a given grid size. For a very large number of processors (e.g. 100 processors in the above case), the grid assigned to a processor becomes very small while the number of faces over which communication takes place becomes larger. The communication time thus becomes larger than the computation time for a given iteration, and the efficiency begins to deteriorate.

Running the same code and the grid on a PC cluster with a large cache memory size further validates the above observation. While each processor of the Origin 2000 has 32KB of cache, each PC CPU has 2 MB of cache memory. This means that even for a lower number of CPUs the cache misses are relatively small, so cache efficiency is not expected to impact as dramatically as on a computer with a small cache memory. Figure 3 shows the performance on a 6-CPU PC cluster. For this figure, a 2-processor time/iteration is taken as baseline.

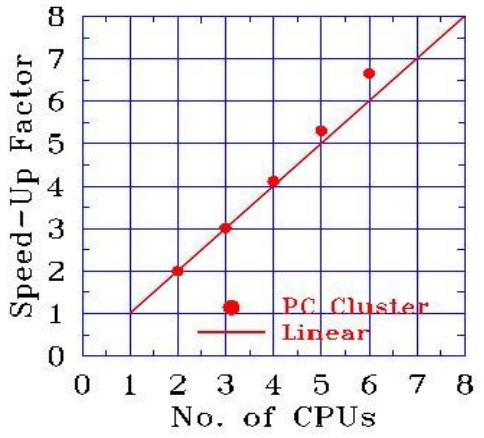

Figure 3 - Speed-Up performance on a PC cluster for ONERA M6 Wing. Mach= 0.8774, α = 5.06 deg. Re_c=11.7 X 10^6.

Comparison with Serial Code

Finally, to demonstrate that no accuracy is lost during parallelization, results from a 100-CPU parallel run are compared to those from a single-CPU serial run. In Figure 4, the normalized residuals and integrated value of lift (C_L) are plotted against the iteration count. A very good agreement between the two confirms the accuracy of the parallel solver relative to the serial version

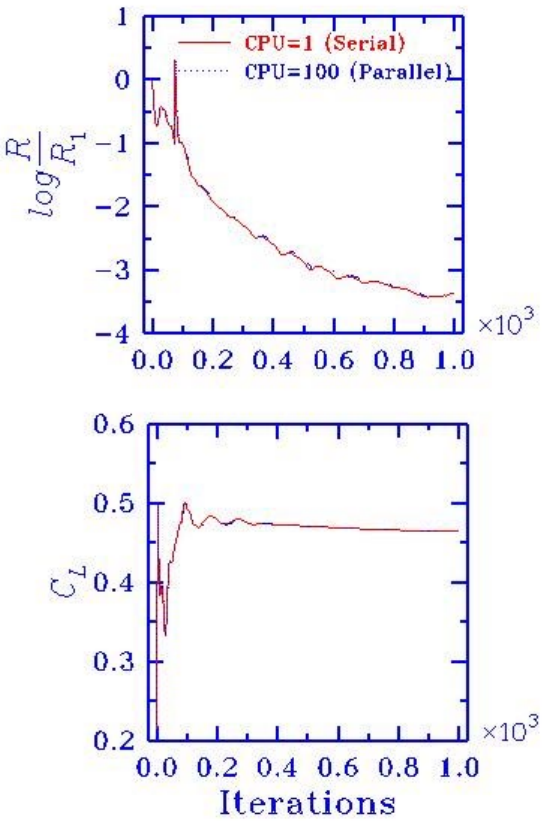

Figure 4 – Comparison of parallel and serial results for ONERA M6 Wing.

F-16 Aircraft with Stores

Parallel, transonic Navier-Stokes computations were performed on a complete F-16 aircraft configuration. A thin-layered, tetrahedral grid, generated for the study reported in Ref. 11, had 1,428,779 cells (255,959 nodes). The normal grid spacing was sized for the wall function to yield a nominal mid-chord $Y^+ \cong 30$ for the first node above the surface and 18 to 20 tetrahedra in the boundary layer. Figure 5 depicts a solid model representation of the

configuration. Although, the full aircraft is shown in the picture, all the computations reported here, were done on a one-half of the symmetric model.

Figure 5 - Solid model depiction of a F-16 aircraft configuration used to evaluate performance of the parallel code.

Figure 6 shows a sample, partitioned grid for this case, divided into 5 partitions; each partition shown with a different color. The parallel solver was run on the Origin 2000 computer system for a Mach number of 0.95, an angle-of-attack of 4.0 deg., and a Reynolds number based on the mean aerodynamic chord (MAC) of 2 million. The efficiency of the parallel code is shown in Figure 7 where the Speed-Up Factor is plotted against the number of CPUs. For this figure, the time/iteration for a 12-CPU run was used as baseline.

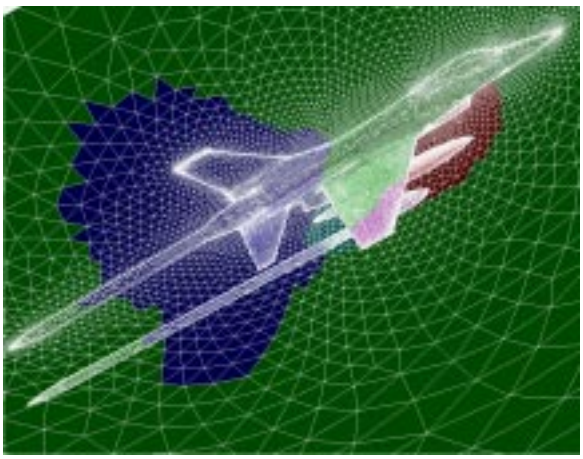

Figure 6 - Computational grid on an F-16 configuration divided into 5 partitions.

Figure 7 shows super-linear performance on this grid. As compared to the ONERA M6 wing (Figure 2), this configuration does not indicate any degradation in speed-up for up to 100 processors. Such a behavior is expected because for the same number of processors, a larger grid still has a larger number of interior cells compared to the inter-partition boundary cells. Thus, communication time can be absorbed within the computational time for a larger number of processors. It is expected that the SF will eventually fall-off as the number of processors is further increased.

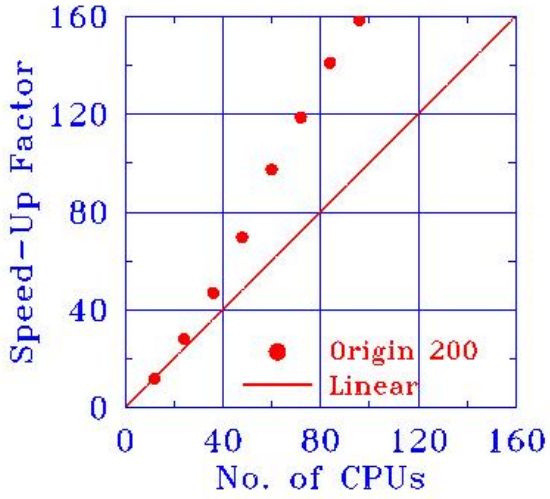

Figure 7. Speed-Up performance on an Origin 2000 computer system. F-16 Configuration. Mach= 0.95, α = 4.0 Deg., Re_c=2.0 X 10^6.

Once again, reproduction of serial results by the parallel code is shown by comparing longitudinal distribution of the pressure coefficient (Cp) along the outboard and inboard sides of the "outer" finned store on the airplane. The comparison, in Figure 8, shows Cp distribution from the serial as well as the parallel codes and is compared against the experimental data from Ref. 12. As can be seen, the serial and the parallel version predict the same behavior attesting to the "faithful reproduction" of the serial results by the parallel code.

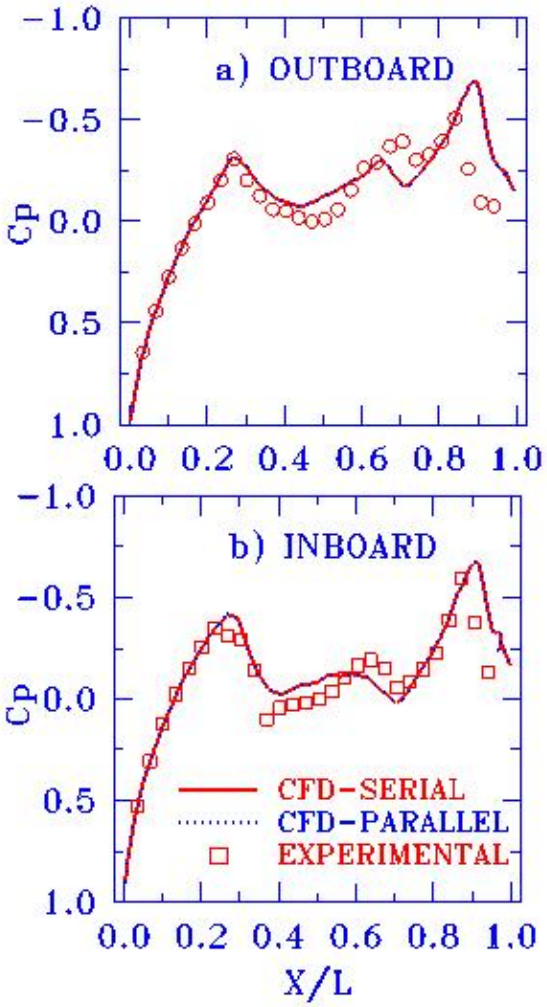

Figure 8. Longitudinal distribution of surface pressure coefficient on the outer "finned-store" on a F-16 configuration.

Turn-Around Time

A major motivation for use of a parallel CFD code is to reduce wall-clock time resulting in rapid turn-around. Development and use of unstructured grids and parallel N-S solvers, like the one described here, is an attempt in this direction. For example the viscous solution on the F-16 configuration, shown earlier, would require about 11.78 hours on single processor of a CRAY C-90. This same case was run in 8.0 hours on a 6-processor PC or in 0.68 hours on a 96-processor Origin 2000.

Tangent-Ogive Forebody

The implementation of the porous boundary condition in USM3Dns [5] has been parallelized presently. The parallel implementation has been on a 5.0 caliber Tangent-Ogive forebody configuration. For computations presented here, only half configuration was modeled. The unstructured grid had 1,009,929 cells and 178,620 points. The grid was suitable for a wall-function application on the solid part of the configuration. The simulated configuration was 4 inches in diameter and 40 inches long. As shown schematically in Figure 9, a surface porosity of 22% was applied on a region from x=1 to 20 inches from the nose.

Figure 9. Surface representation of 5.0 caliber tangent-ogive configuration. Porous region indicated in red.

Navier-Stokes solutions using the serial and the parallel codes, were obtained on this configuration at Mach number = 0.3, Angle-of-attack of 30 degrees and a Reynolds number based on the diameter of 0.4 million. A solution with porosity turned-off was obtained first. Then the porosity was turned-on and the solution restarted. The porous solution was terminated when the plenum pressure converged to a steady state value. Figure 10 shows surface Cp contours for the solid (a) as well as the porous (b) conditions, and clearly shows the effect of porosity in diffusing the pressure gradients on the porous part of the configuration.

a)

b)

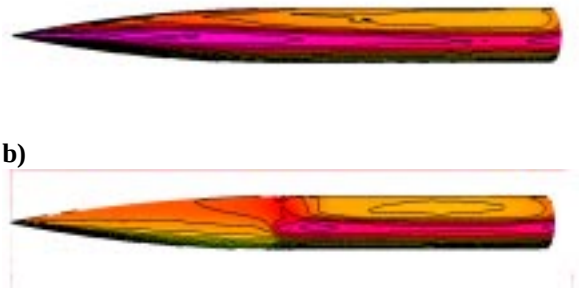

Figure 10. Surface Cp on 5.0 caliber tangent-ogive configuration. a) Solid, b) Porous.

Figure 11, shows comparison of surface Cp distribution between computed and experimental data taken from Ref. 13, at a streamwise station 10 inches from the nose. The windward centerline

corresponds to a φ = 0 deg. and its leeward counterpart to φ = 180 deg. The curves representing the single and parallel solutions compare reasonably well with each other, although both the computed solutions do not exactly match with the experimental data

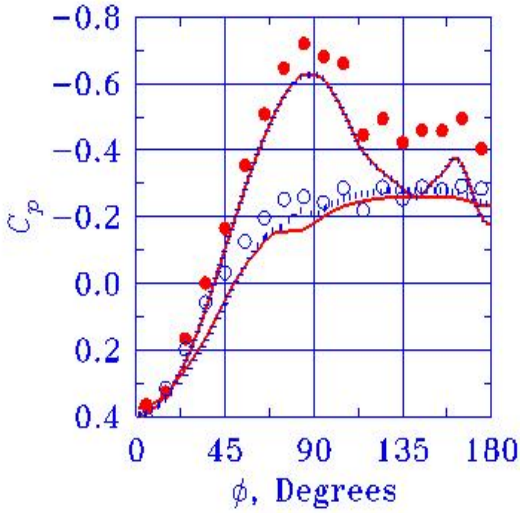

Figure11. – Comparison of circumferential Cp at x/D=2.5. (Lines USM3Dns, Symbols – Experimental data, Red = Solid, Blue = porous) .

SUMMARY

A parallel version of an unstructured-grid based Navier-Stokes solver, previously developed for operation on both shared as well as distributed memory parallel computing systems, has been enhanced to include upgrades made to the serial version of the code. Scalability of the resultant parallel code is demonstrated by example flow solutions on three configurations of aerospace interest. An innovative grid renumbering construct and overlapping communication and computational times during flow iterations is shown to produce linear or super linear performance. Fast parallel solvers like the one reported here along with affordable PC clusters are sure to enhance productivity of future engineering design cycles.

REFERENCES

1. Bhat, M. K. and Parikh, P. C., "Parallel Implementation of an Unstructured Grid-Based Navier-Stokes Solver", AIAA Paper 99-0663, January 1999.
2. Grismer, M.J., Strang, W.Z., Tomaro, R.F. and Witzeman, F.C., "A Parallel, Implicit, Unstructured Euler/Navier-Stokes Solver", AIAA Paper 97-0333, 1997.
3. Mavriplis, D. J., "Three-Dimensional High-Lift Analysis Using a Parallel, Unstructured Multigrid Solver", AIAA-98-2619-CP.
4. Frink, N. T., Pirzadeh, S., Parikh, P. C., Pandya, M. J. and Bhat, M. K., " The NASA Tetrahedral Unstructured Software System (TetrUSS)", *The Aeronautical Journal*, Vol. 104, No. 1040, October 2000, pp. 491-499.
5. Frink, N.T., Bonhaus, D.L., Vatsa, V.N., Bauer, S.X.S, and Tinnetti, A., "A Boundary Condition for Simulation of Flow Over Porous Surfaces", AIAA Paper 2001-2412, June 2001.
6. Frink, N. T., " Tetrahedral Unstructured Navier-Stokes Method for Turbulent Flows", *AIAA Journal*, Vol. 36, No. 11, pp.1975-1982, November 1998,.
7. Roe, P.L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", *J. Comp. Physics.*, Vol. 43, No. 2, pp. 357-372, 1981.
8. Spalart, P.R., and Allmaras, S.R., "A One-Equation Turbulence Model for Aerodynamic Flows", AIAA Paper 92-0439, 1992.
9. Karypis, G. and Kumar, V., "METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Ordering of Sparse Metrices", Version 3.0.3, University of Minnesota, November 1997.
10. Kumar, V., Grama, A., Gupta, A. and Karypis, G., "*Introduction to Parallel Computing – Design and Analysis of Algorithms*", The Benjamin Cummings Publishing Company, Inc., 1994.
11. Frink, N.T. and Pirzadeh, S., "Tetrahedral Finite-Volume Solutions to the Navier-Stokes Equations on Complex Configurations", *Int. J. of Numerical Methods in Fluids*, Volume 31, pp. 175-187, 1999.
12. Hoph, J., "Separation Characteristics of the DWS-24 Dispenser, the MK-84 LDGP, the 370-GAL Tank (E)+Pylon, and the Generic Missile (Metric and Pressure Instrumented) in the Flow Field of the F-16 Aircraft", AEDC-TSR-94-1, February 1994.
13. Bauer, S.X.S. and Hemsch, M.J.," Alleviation of Side Force on Tangent-Ogive Forebodies Using Passive Porosity", *Journal of Aircraft*, No. 2, pp. 354-361, 1993.